Unidad II: Programación orientada a objetos básica

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021

Agenda

1. **Introducción al paradigma orientado a objetos**
   - ¿Por qué utilizar POO?
   - Abstracción: caracterización de la realidad en el paradigma orientado a objetos
   - Conceptos fundamentales de la POO
2. **Uso de clases y objetos**
   - Ejemplo: clase Cocina
   - Variables de tipo Objeto
3. **Diseño de una clase: 7 pasos a seguir**

1. Introducción al paradigma orientado a objetos

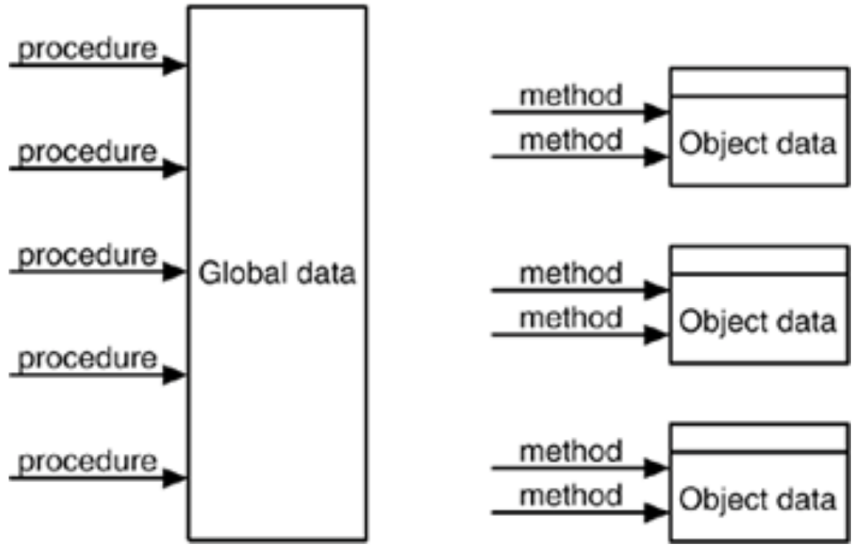
Paradigma de programación

Un paradigma es una forma de entender y representar la realidad: un conjunto de teorías, estándares y métodos que, juntos, representan un modo de organizar el pensamiento, es decir, **un modo de ver el mundo**. Cada nuevo paradigma responde a una necesidad real de nuevas formas de afrontar problemas. A menudo un nuevo paradigma es creado como respuesta a las deficiencias de paradigmas anteriores.

La programación orientada a objetos es una de los actuales paradigmas predominantes en el desarrollo de software, habiendo reemplazado al paradigma **estructurado** que fue desarrollado en la década de los 70's.

1.1 ¿Por qué utilizar POO?

Tradicionalmente la programación estructurada consiste en el diseño de un conjunto de funciones que resolverán algún problema. Para problemas **pequeños**, la subdivisión funciona muy bien. Pero los objetos son más apropiados para problemas más **grandes**.



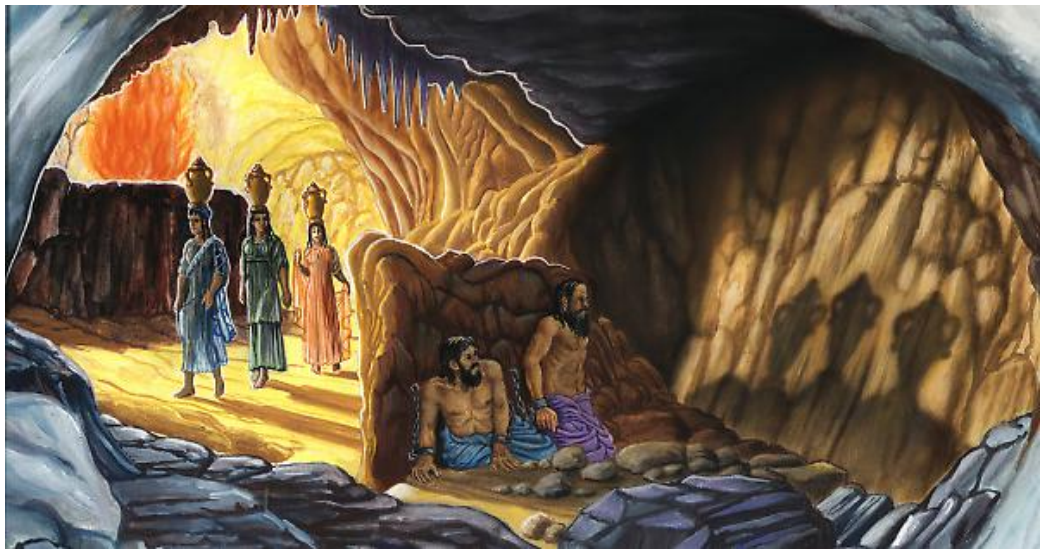
Por ejemplo un sistema que requiera 2,000 funciones, en un estilo orientado a objetos podría organizarse en 100 clases con un promedio de 20 funciones cada una. Este enfoque será mucho más fácil de desarrollar, ya que al buscar un error es más sencillo hacerlo en una clase con 20 funciones que en un solo programa estructurado con 2,000 de ellas.

1.2 Abstracción: caracterización de la realidad en el paradigma orientado a objetos

Antes de continuar, hay que resolver las siguientes preguntas claves:

- **¿Qué es una clase?**
- **¿Qué es un objeto?**
- ¿Cuál es la diferencia entre estos conceptos?

La respuesta la podemos encontrar en la filosofía, sí, no es broma xD.

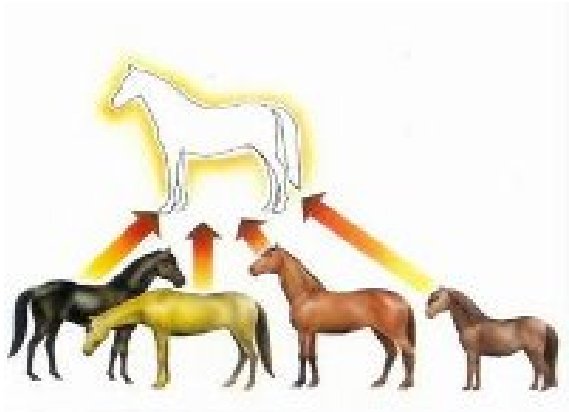


1.2 Abstracción: caracterización de la realidad en el paradigma orientado a objetos

Teoría de las ideas de Platón, según él existen dos mundos:

1. **Mundo Inteligible o de las Ideas** es la auténtica realidad, el lugar donde habitan las **Ideas perfectas**: a este mundo no se puede acceder con el uso de los sentidos sino mediante la razón.
2. **Mundo Sensible o Visible:** es el conjunto de todo aquello que se muestra a los sentidos, las cosas físicas que se caracterizan por ser temporales, cambiantes, corruptibles e **imperfectas**.

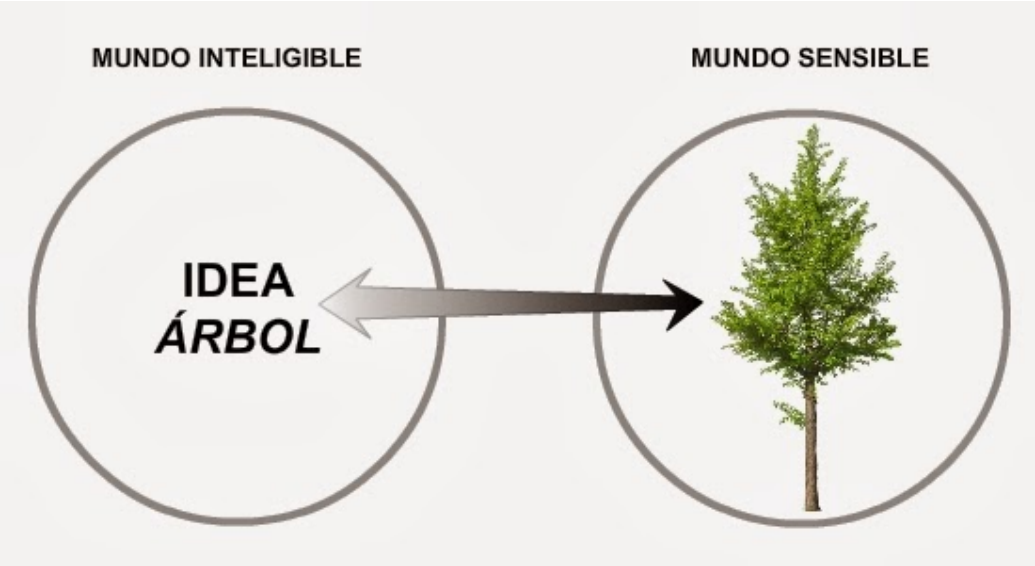
1. **Idea de caballo:** "molde" que describe a todos los caballos que existen, concepto ideal y eterno.
2. **Caballos en particular:** nacen, viven, mueren, tienen un nombre en específico, una raza en específico, todos son diferentes.



1.2 Abstracción: caracterización de la realidad en el paradigma orientado a objetos

Ahora podemos dar respuesta a estas preguntas:

- **¿Qué es una clase?** Molde de un árbol genérico, listado de cosas que describen a **todos** los árboles y listado de acciones que **todos** realizan.
- **¿Qué es un objeto?** Un árbol en específico, ejemplos: un árbol de **eucalipto** de 2 metros de altura, un **roble** de 10 metros, etc.



¿Cuál es la diferencia?
Las clases se programan en archivos aparte y los objetos se utilizan en el **Main**.

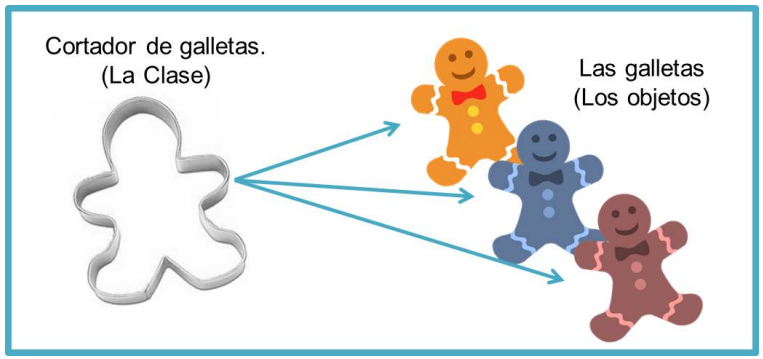
1.3 Conceptos fundamentales de la POO

Pongamos un ejemplo: un automóvil.

- ¿Qué características/propiedades/atributos tiene un automóvil?
  - Color.
  - Marca y modelo. $\Longrightarrow$ Estos se conocen como atributos.
  - Capacidad del tanque de gasolina.
- ¿Qué acciones puede realizar un automóvil?
  - Encender y apagar.
  - Mover. $\Longrightarrow$ Estos se conocen como métodos.
  - Echar gasolina.

1.3 Conceptos fundamentales de la POO

- **Clase:** Molde genérico (relacionado al mundo inteligible de Platón).
  Ejemplo: Persona{nombre, edad y género}.
- **Objeto:** Caso particular de una clase (relacionado al mundo sensible de Platón), contiene características específicas.
  Ejemplo: persona01{"ronaldo", 26, masculino}.



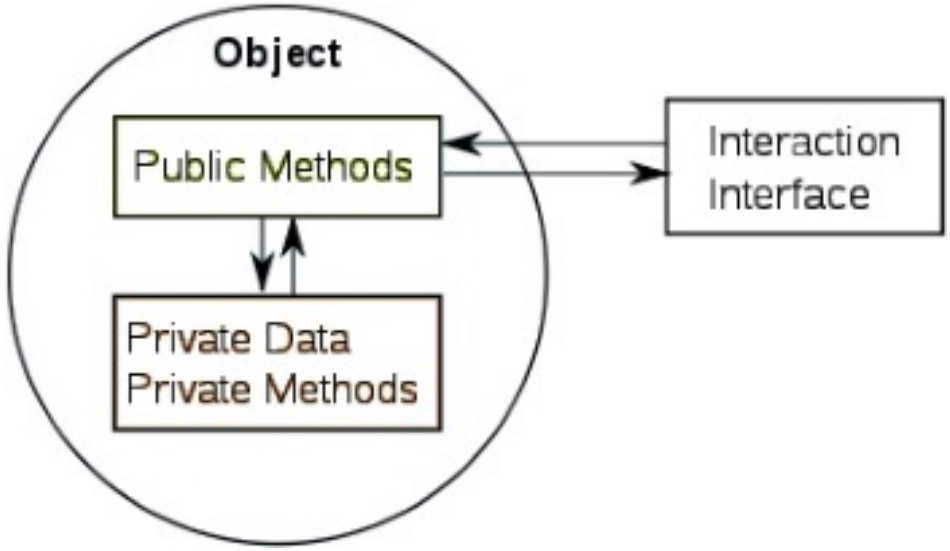
- **Atributo:** Característica o propiedad.
- **Método:** Acción o función.
- **Instanciación:** Proceso de creación de un objeto a partir de una clase. Cuando se dice que un objeto es una instancia de una clase, se refiere a que un objeto es un caso específico o particular de una clase.

2. Uso de clases y objetos

Es correcto imaginar los objetos como **cajas negras** que constan de:

- Una **interfaz pública** (los métodos que podemos llamar).
- Una **implementación** oculta (el código fuente necesario para hacer que dichos métodos funcionen).

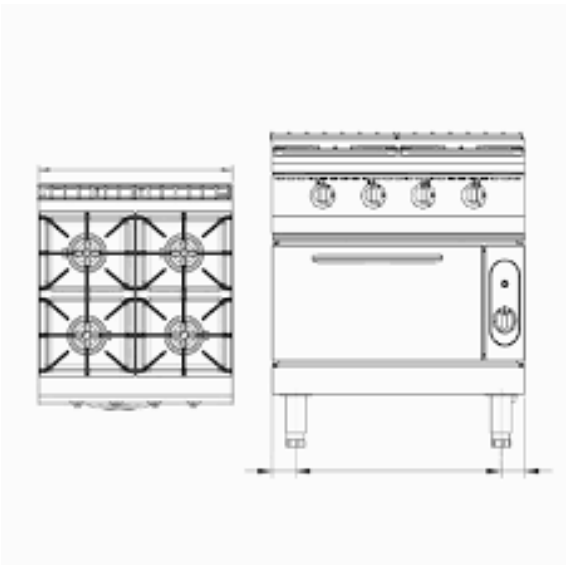
La **encapsulación** es un principio fundamental de la POO y consiste en ocultar el estado interno del objeto y obligar a que toda interacción se realice a través de los métodos (públicos) del objeto.



2.1 Ejemplo: clase Cocina

Ejemplo: clase Cocina

¿Qué características definen a todas las cocinas? ¿Qué acciones realizan todas las cocinas? ¿Qué atributos y métodos debe tener una clase Cocina?



Clase: Cocina

Objeto: cocina01

Objeto: cocina02

2.1 Ejemplo: clase Cocina

Por ejemplo, podríamos querer crear (instanciar) diferentes cocinas de la siguiente manera:

```
public class POObasica {
    public static void Main() {
        Cocina cocina01 = new Cocina(4, "verde", false);
        Cocina cocina02 = new Cocina(6, "gris", true);
    }
}
```

El operador **new** provoca la instanciación del objeto *cocina01* de la clase Cocina. Dicho proceso es llamado **construcción**. Los valores: 4, verde y false; son llamados **parámetros de construcción**.

2.1 Ejemplo: clase Cocina

Para poder realizar lo anterior, es necesario implementar un **constructor** a la definición de la clase. ¿Qué hace un constructor? **Especifica cómo deberá ser inicializado un objeto.** En nuestro ejemplo solamente se tienen tres parámetros de construcción: cantidad quemadores, color y una bandera que indique si se posee horno o no.

Constructores

Siempre deben tener el mismo nombre de la clase cuyos objetos está construyendo. De manera similar a los métodos, los constructores se declaran como públicos para permitir poder construir objetos desde cualquier método de nuestra aplicación. La principal diferencia de un constructor con los métodos es que **los constructores no tienen un tipo de retorno**.

2.2 Variables de tipo Objeto

Una **variable** es un elemento de información que se encuentra alojado en la memoria de la computadora. Dicha ubicación (podríamos imaginarla como una casilla) se identifica por un nombre simbólico, es decir, el nombre de la variable. En los lenguajes de programación orientados a objetos, el tipo de dato de una variable puede ser uno de dos posibilidades:

- **Tipo de datos primitivo** (por ejemplo int, char, float, double, bool). En este caso, se almacena el valor *crudo* en la casilla de la memoria.
- **Variable de tipo objeto** (por ejemplo Cocina, Rectangle). En este caso, en la casilla de memoria se almacena **la dirección de memoria del objeto**[a].

[a] Concepto de puntero.

2.2 Variables de tipo Objeto

Es importante distinguir los siguientes momentos:

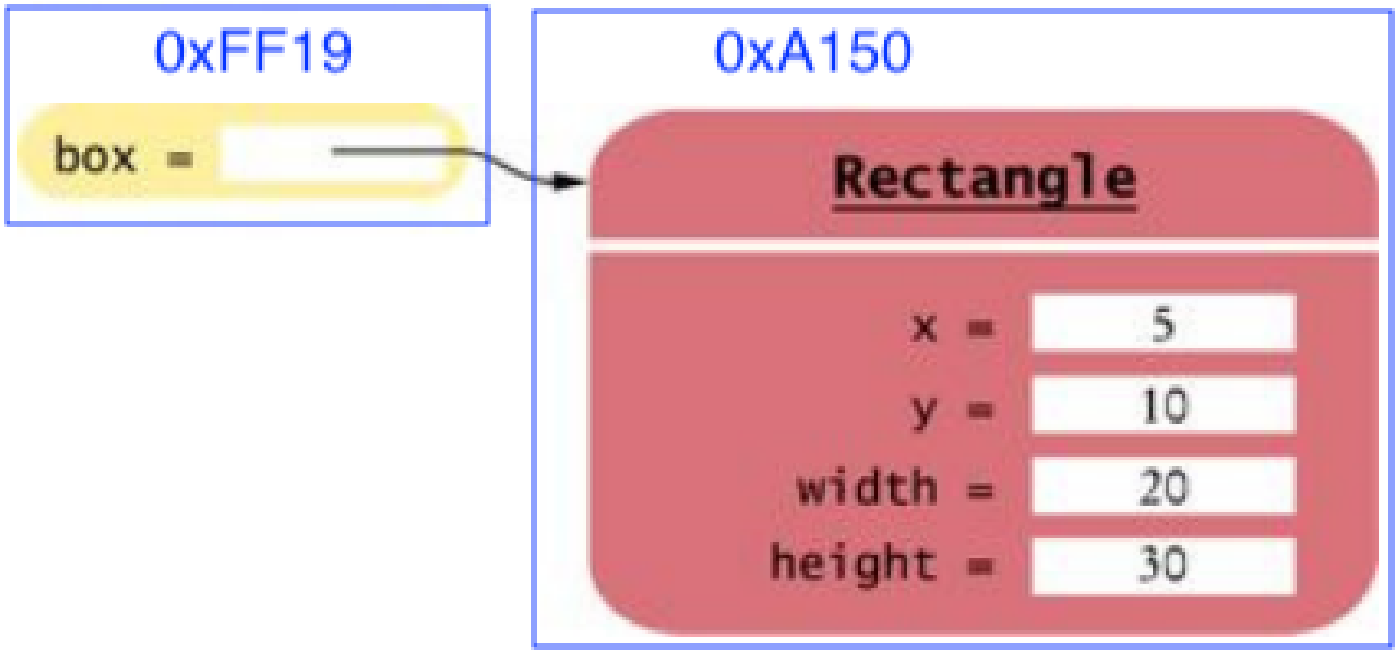
1. **Declaración de la variable tipo objeto:** se crea una casilla en la memoria (por ejemplo, en la dirección 0xFF19), no se almacena ningún valor en ella. Dicha casilla se identifica por el nombre "cocina01".
2. **Instanciación del objeto:** se crea un espacio en la memoria destinado a almacenar todo lo que contiene dicho objeto (por ejemplo, en la dirección 0xA150). Esto se realiza con el operador **new**.
3. **Construcción del objeto:** se utilizan los parámetros de construcción (se almacenan a partir de la dirección 0xA150).
4. Se **almacena la dirección del objeto** (0xA150) **en la casilla de la variable** de tipo objeto (0xFF19). En otras palabras: en la casilla 0xFF19 se almacena el número 0xA150.

Cocina cocina01 = new Cocina (4, "verde", false);

2.2 Variables de tipo Objeto

Resumiendo:

- Declaración de la variable tipo objeto: casilla 0xFF19.
- Instanciación del objeto: reserva casilla 0xA150 con **new**.
- Construcción del objeto: guardar valores en dirección 0xA150.
- Almacenar[1] la dirección 0xA150 dentro de la casilla 0xFF19.



[1] Concepto de puntero.

3. Diseño de una clase: 7 pasos a seguir

Paso 1: Pensar qué es lo que se necesita hacer con un objeto de la clase que se va a crear

Por ejemplo, para crear una clase llamada **Automóvil** no es necesario modelar todas y cada una de las características y acciones de un automóvil de la vida real (hay demasiadas). Al momento de crear una clase, se debe tener una idea de qué se va a hacer con ella. Se recomienda escribir en un papel qué acciones se deberán ejecutar. Por ejemplo:

- Poner gasolina.
- Avanzar cierta distancia.
- Consultar cuánta gasolina queda en el tanque.

3. Diseño de una clase: 7 pasos a seguir

Paso 2: Encontrar nombres para los métodos

Buscar nombres apropiados para los métodos y aplicarlos a un objeto de ejemplo (imaginar que ya se ha programado la clase).

```
Carro unCarro = new Carro(...);

unCarro.ponerGasolina(20);
unCarro.avanzar(100);
unCarro.consultarGasolina();
```

Notar que todavía no se tienen todos los detalles, por ejemplo los parámetros de construcción.

3. Diseño de una clase: 7 pasos a seguir

Paso 3: Documentar la interfaz pública

Comentar, de manera sencilla, qué hace la clase y c/u de sus métodos.

```
//Un Carro que puede avanzar y consumir gasolina.
public class Carro {
    //Añadir gasolina al tanque. Cantidad se mide en galones.
    public void ponerGasolina(double cantidad) { }
    //Avanzar cierta distancia (kilómetros), consumiendo gasolina.
    public void avanzar(double distancia) { }
    //Obtener la cantidad de galones de gasolina restantes en el tanque.
    public double consultarGasolina() { }
}
```

Todavía no se sabe cómo funcionarán los métodos, ahorita están vacíos.

3. Diseño de una clase: 7 pasos a seguir

Paso 4: Determinar qué atributos son necesarios

Pensar qué datos son necesarios conocer para poder realizar las acciones que ya contemplamos en pasos anteriores. Es importante recordar que **los métodos pueden ser ejecutados en cualquier orden**. Se recomienda elegir un método en específico (el más sencillo o el más interesante) y pensar qué datos se necesitan. En el ejemplo de la clase Carro, tendría sentido almacenar un atributo llamado **gasolina**.

```
public class Carro {
    private double gasolina;
    ...
}
```

3. Diseño de una clase: 7 pasos a seguir

Ahora, el método **ponerGasolina** simplemente le suma una cantidad al atributo gasolina. El método **avanzar** debe reducir la cantidad de gasolina en el tanque. Pero, ¿qué tanto? Eso depende de la eficiencia del automóvil. Por ejemplo: si se quiere conducir 100 kilómetros y el carro tiene una eficiencia de 20 kilómetros por galón, entonces se necesitarán 5 galones de gasolina para poder ejecutar dicha acción. Candidato perfecto para ser un atributo.

```
public class Carro {
    private double gasolina;
    private double eficiencia;
    ...
}
```

3. Diseño de una clase: 7 pasos a seguir

Paso 5: Determinar los constructores

Es común solicitar como parámetros de construcción valores para todos los atributos de nuestra clase, pero es correcto hacer la siguiente pregunta: ¿cuáles datos son esenciales?

También es común poseer dos constructores: uno que inicialice todos los atributos con valores por defecto (cero, por ejemplo) y otro que los inicialice con valores provistos por el usuario.

Para el ejemplo de la clase Carro, podría ser correcto iniciar con un tanque de gasolina vacío. Pero es esencial conocer la eficiencia de dicho carro, es decir, no hay ningún valor por defecto para la eficiencia de un carro genérico. Por tanto debe ser considerado como un parámetro de construcción.

3. Diseño de una clase: 7 pasos a seguir

Paso 6: Implementar los métodos

Implementar los métodos, uno por uno, comenzando por los más sencillos. Si se tiene dificultades con la implementación, es posible que sea necesario regresar un par de pasos. Es común no tomar en cuenta algún factor en los pasos anteriores.

Paso 7: Probar la clase (utilizar su interfaz pública desde el main)

Digitar un pequeño programa de prueba. Dicho programa podría ser el mismo que se ideó en el paso 2.

```
Carro unCarro = new Carro(20);

unCarro.ponerGasolina(20);
unCarro.avanzar(100);
```

Unidad II: Programación orientada a objetos básica

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021